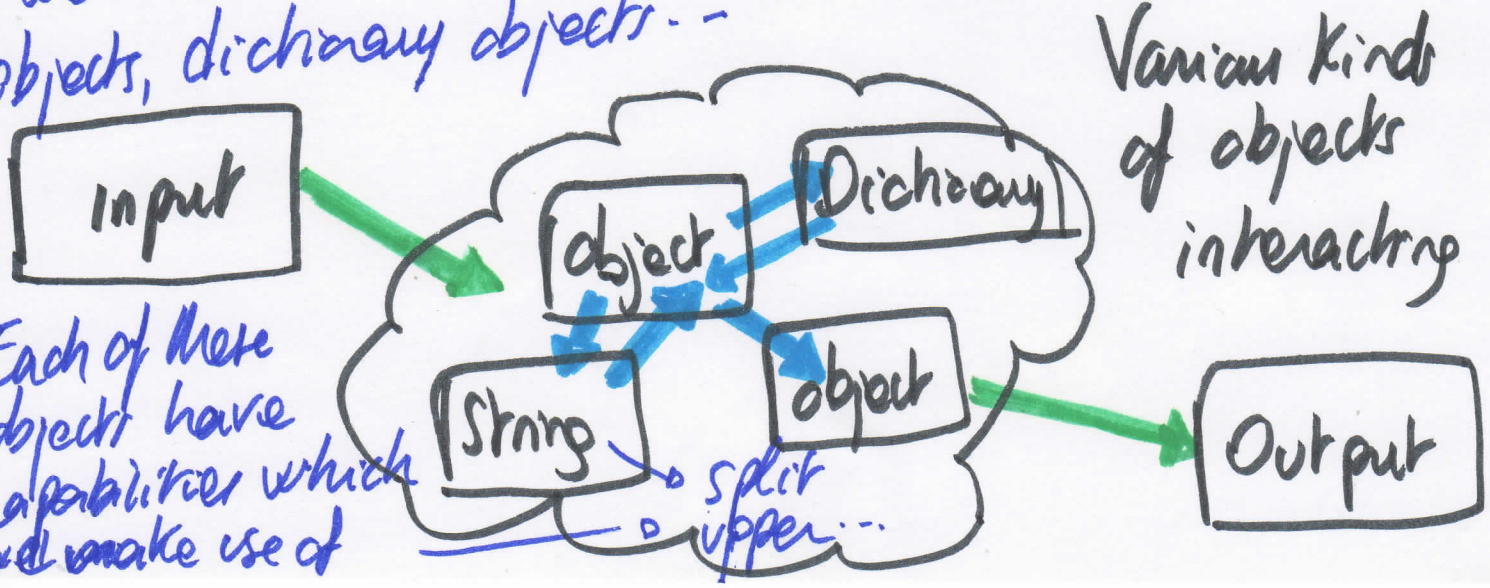
(58) Using databases with Python — Uni of Michi chapter (14)

Object oriented programming [lecture to be familiar with the terms]

- Methods are like functions that are part of an object
  E.g list.append() ← method
- A program is made up of many cooperating objects.
- Instead of being the "whole program" - each object is a little "island" within the program and cooperatively working with other objects.
- A program is made up of one or more objects working together - objects make use of each other's capabilities.

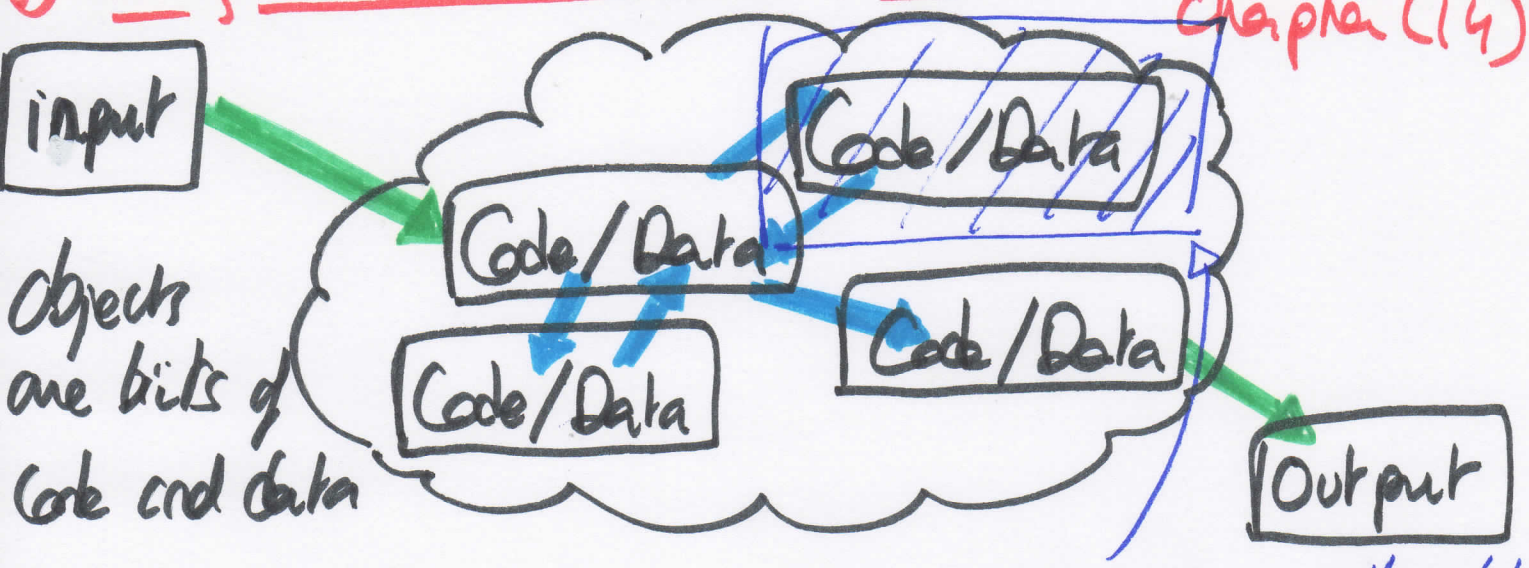
Object

- An object is a bit of self-contained Code and Data
- A key aspect of the Object approach is to break the problem into smaller understadable parts (divide and conquer)
- Objects have boundaries that allow us to ignore unneeded detail.
- We have been using objects all along: String Objects, integer objects, dictionary objects...



69 Using databases with Python

Univ of Michigan chapter (14)



- Objects hide detail. They allow us to ignore the detail of the 'rest of the program'. It hides the complexity of the program. The complexity can be contained in objects, making the program easier to deal with.

Terminology

- **Class**: a template - Dog
- **Method** or **Message**: A defined capability of a class bark()
- **Field** or **Attribute**: A bit of data in a class - length
- **Object** or **Instance**: A particular instance of a class - lassie

Terminology: Class

Defines the abstract characteristics of a thing (object), including the thing's characteristics (its attributes, fields or properties) and the thing's behaviors (the things it can do, or methods, operations or features).

(60) Using databases with Python — Univ of Michigan Chapter (14)

One might say that a **class** is a **blueprint** or factory that describes the nature of something. For example the class Dog would consist of traits shared by all dogs such as breed and fur color (characteristics) and the ability to bark and sit (behaviors).

A pattern of a class. The class of Dog defines all possible dogs by listing the characteristics and behaviors they can have; the object Lassie is one particular dog, with particular version of the characteristics. A dog has fur; Lassie has brown- and white fur.

Terminology: Instance

- One can have an **instance** of a class or a particular object. The instance is the actual object created at runtime. In programmer jargon, the Lassie object is an instance of the Dog class. The set of values of the attributes of a particular object is called its state. The object consists of state and the behaviors that is defined in the object's class.

Object and Instance are often used interchangeably

Terminology: Method

An object's abilities. In language, methods are verbs. Lassie, being a Dog, has the ability to bark. So bark() is one of Lassie's methods. She may have other methods as well, for example eat(), eat().

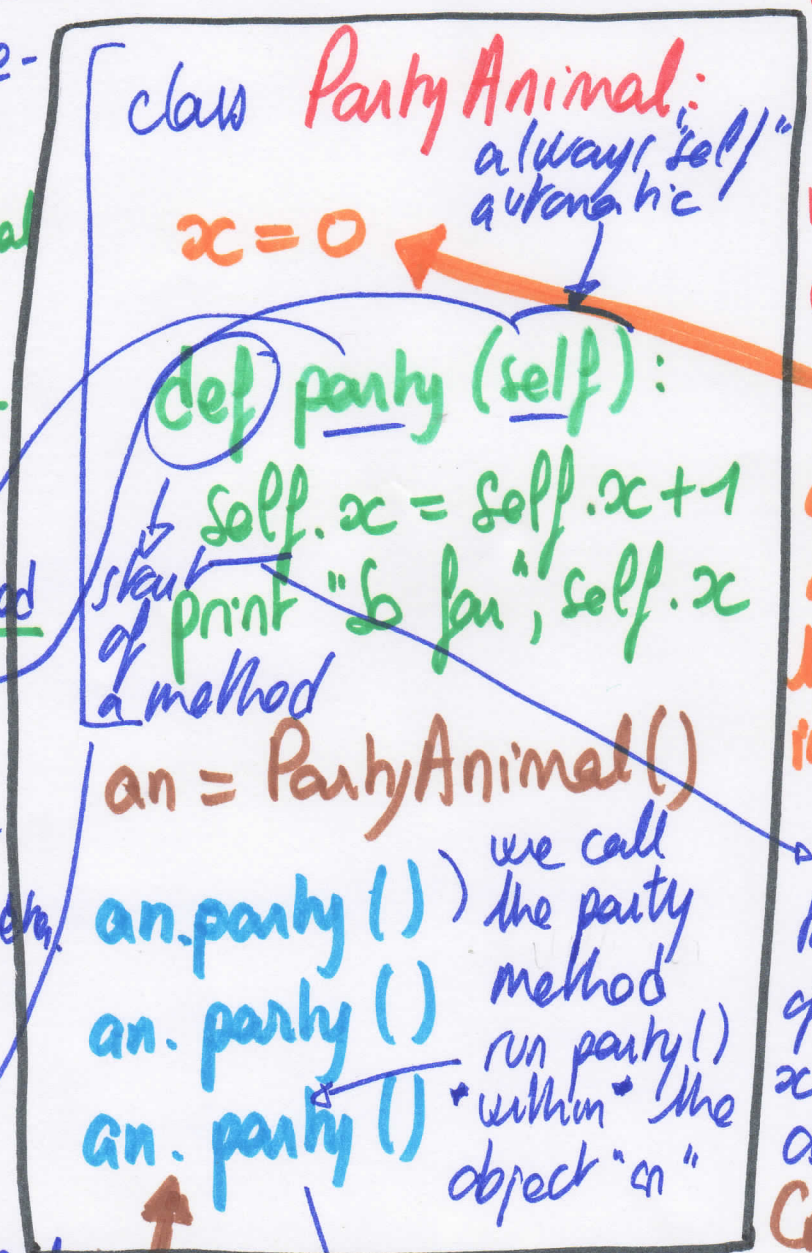
(61) Using databases with Python

Uni of Michigan chapter (14)

Within the program, using a **method** usually affects only one particular object; all Dogs can bark, but you need only one particular dog to do the barking.

Method and **Message** are often used interchangeably

Simple Python object - Object oriented Python



class is a reserved word

Each PartyAnimal object has a bit of code (indented)

name of the method parameter.

Every method has to have at least one parameter. If more, we put after self.

All recorded within PartyAnimal

start of a method

always "self" automatic

This is the template for making PartyAnimal objects (indented)

Each PartyAnimal object has a bit of data. Variable that is part of any instance of this class

It is a way to talk within an instance of an object to the x that is part of the object

we call the party method

run party() "within" the object "an"

Create a PartyAnimal object (called an)

Tell the object to run the party() code.

PartyAnimal.party(an) = an.party()

We can do things to the "an" object based on the definition of the class

(62) Using database with Python — Univ of Michigan chapter (14)

an
self x = 0
party()

⇨ an = PartyAnimal()
variable "an" with data and code
x = 0 party()

output

So far 1 ⇨ an.party()
So far 2 ⇨ an.party()
So far 3 ⇨ an.party()

⇨

```
def party(self):
  self.x = self.x+1
  print "So far", self.x
```

- "self" is a formal argument that refers to the object itself
- self.x is saying "x within self"
- self is "global within this object"
- self is a way to access the [code] instance of the object

What we have done with "class" is creating another type().

A handy way to find capabilities

```
x = list()
type(x) <type 'list'>
dir(x)
```

The dir() command lists capabilities.

(63) Using databases with Python

Uni of Michigan chapter (14)

['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', ..., 'append', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']

↳ methods

- Ignore the ones with underscores - these are used by Python itself
- The rest are real operations that the object can perform
- It is like type() - it tells us something *about* a variable

- We can use dir() to find the "capabilities" of our newly created class.

```
print "Type", type(an)
print "Dir", dir(an)
```

instance of an object / class

```
Type <type 'instance'>
Dir ['__doc__', '__module__', 'party', 'x']
```

Object Life Cycle

- Object are created, used and discarded
- We have special blocks of code (methods) that get called
  - At the moment of creation (constructor)
  - At the moment of destruction (destructor)
- Constructors are used a lot
- Destructors are seldom used

(64) Using database with Python — Univ of Michi chapter (14)

Constructor

- The primary purpose of the constructor is to set up some instance variables to have the proper initial values when the object is created.
- In **object-oriented programming**, a **constructor** in a class is a special block of statements called when an object is created.

```
class PartyAnimal:
    x = 0
    def __init__(self):
        print "I am constructed"
    def party(self):
        self.x = self.x + 1
        print "So far", self.x
    def __del__(self):
        print "I am destructed", self.x
an = PartyAnimal()
an.party()
an.party()
an.party()
```

very common

very rare

let's construct the PartyAnimal

Output

```
I am constructed
So far 1
So far 2
So far 3
I am destructed 3
```

- The constructor and destructor are optional. The constructor is typically used to set up variables. The destructor is seldom used.

(68) Using databases with Python — Univ of Michigan chapter (14)

Many instances

- We can create **lots of objects**. the class is the template for the object
- We can store each **distinct object** in its own variable.
- We can have multiple **instances** of the same class
- We call this having multiple **instances** of the same class
- Each **instance** has its own copy of the **instance variables**

```
class PartyAnimal:
    x = 0
    name = ""
    def __init__(self, nam):
        self.name = nam
        print self.name, "constructed"
    def party(self):
        self.x = self.x + 1
        print self.name, "party count", self.x
```

added by Python

Constructors can have additional **parameters**. These can be used to setup **instance variables** for the particular instance of the class (i.e. for the particular object).

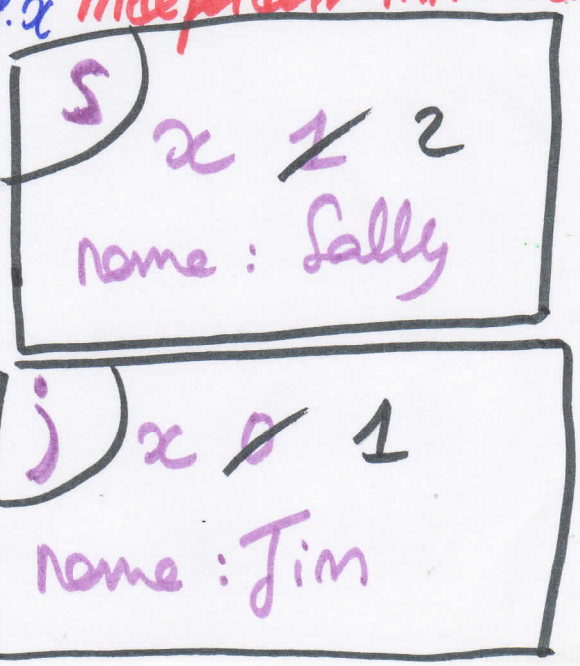
Run this code but doesn't do anything else than recording it

We have two independent instances

```
s = PartyAnimal("Sally")
s.party() 1
j = PartyAnimal("Jim")
j.party() 1
s.party() 2
```

s = an object ⇒ instance 1

j = an object ⇒ instance 2



(66) Using databases with Python

Univ of Michigan
Chapter (14)

Inheritance (terminology)

- When we make a new class, we can reuse an existing class and inherit all the capabilities of an existing class and then add our own little bit to make our new class
- Another form of store and reuse
- Write once - reuse many times
- The new class (child) has all the capabilities of the old class (parent) - and then some more.
- "Subclasses" are more specialized versions of a class, which inherit attributes and behaviors from their parent classes, and can introduce their own.

FootballFan is a class which extends PartyAnimal It has all the capabilities of PartyAnimal and more.

```
class PartyAnimal:
  x = 0 . . .
  name =
  def __init__(self, nam):
    self.name = nam
    print self.name, "constructed"
  def party(self):
    self.x = self.x + 1
    print self.name, "party count", self.x

class FootballFan(PartyAnimal)
  points = 0
```

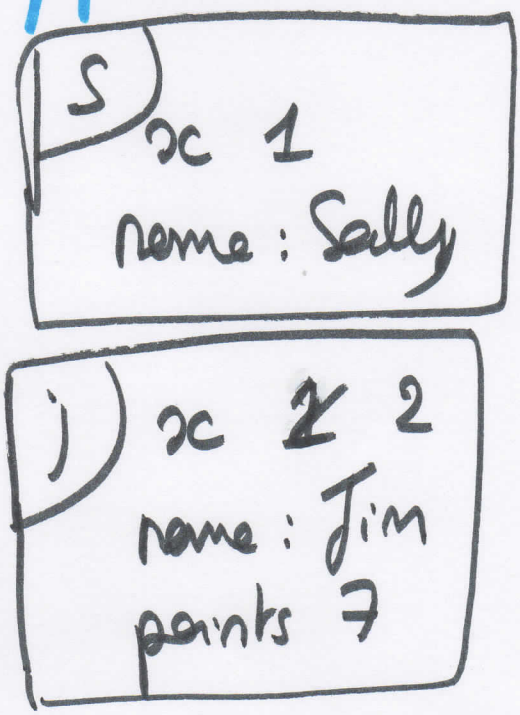
automatically here

(7) Using databases with Python

Univ of Michigan
Chapter (16)

```
def touchdown(self):
    self.points = self.points +7
    self.party()          → new method
    print self.name, "points", self.points

s = PartyAnimal("Sally")
s.party()
j = FootballFan("Jim")
j.party()
j.touchdown()
```



Object oriented programming summary

- Object oriented programming is a very structured approach to code reuse.
- We can group data and functionality together and create many independent instances of a class.

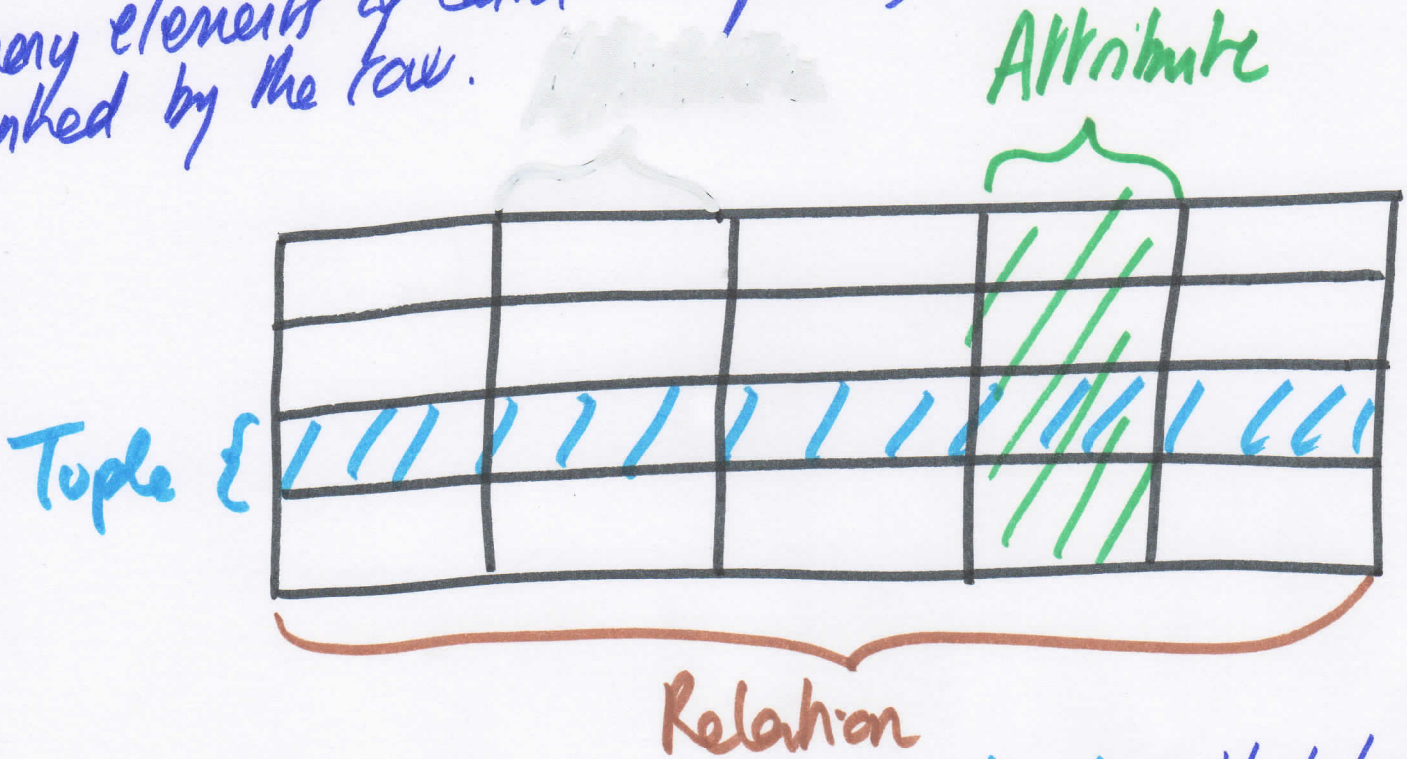
Relational databases

- Relational databases model data by storing rows and columns in tables. The power of the relational database lies in its ability to efficiently retrieve data from those tables and in particular where there are multiple tables and the relationships between these tables involved in the query.

(68) Using databases with Python — Univ of Michigan, Chapter 16

Terminology - Relational databases

- **Database**: contains many tables
- **Relation (or table)**: contains tuples and attributes
- **Tuple (or row)**: a set of fields that generally represents an "object" like a person or a music track
- **Attribute (also column or field)**: one of possibly many elements of data corresponding to the object represented by the row.



- A **relation** is defined as a **set of tuples** that have the **same attributes**. A **tuple** usually represents an **object** and information about that object. Objects are typically physical objects or concepts. A **relation** is usually described as a **table**, which is organized into **rows** and **columns**. All the data referenced by an **attribute** are in the **same domain** and conform to the **same constraints**.

(69) Using databases with Python

Uni of Michi
Chapter 14

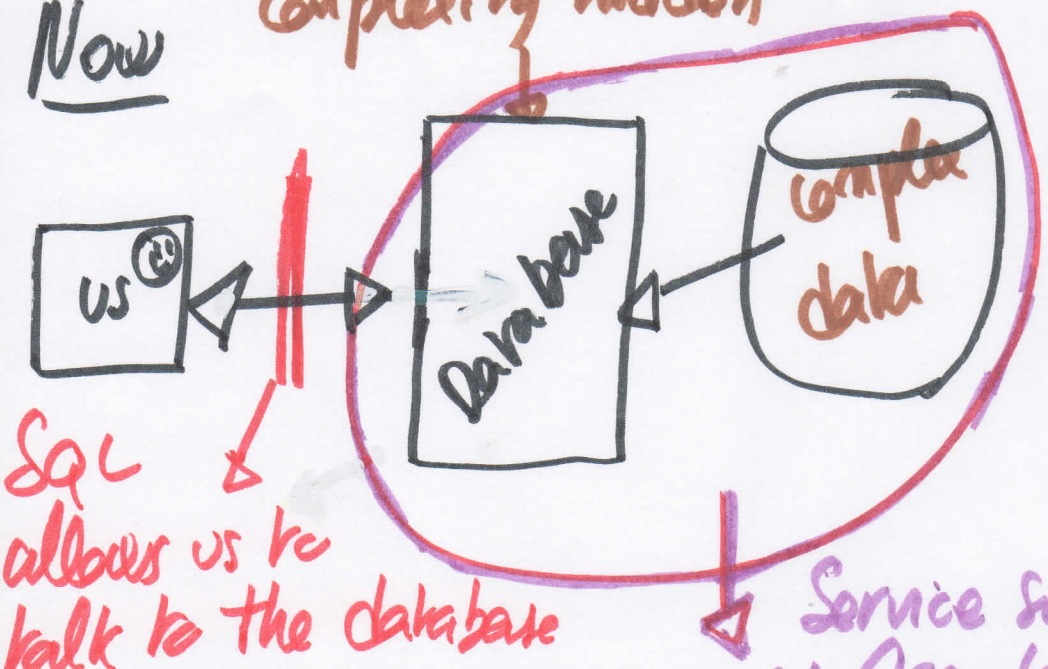
SQL - Structured Query Language

- It is the language we use to issue commands to the database
  - Create a table
  - Retrieve some data
  - Insert data
  - Update data
  - Delete data

60's 70's

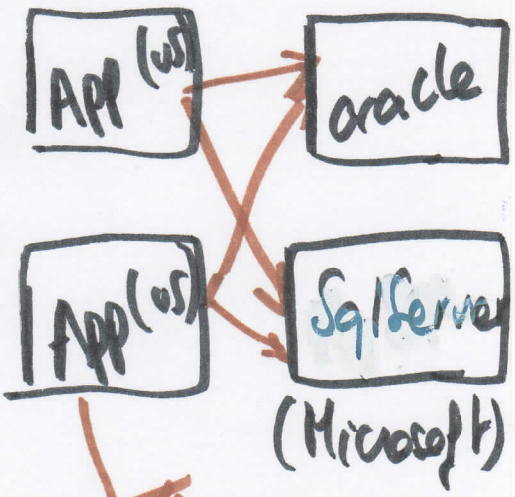
- Data was represented in a disk
- They found ways to access data more rapidly

Now



SQL allows us to talk to the database

Service such as Oracle



The app can talk to any thanks to SQL

- SQL makes the communicate to databases portable.
- Python clean up the data (SQL cannot handle messy data, it needs 'clean data')
- SQL is then amazing to store and retrieve data

(40) Using databases with Python

Univ of Michigan
Chapter 14

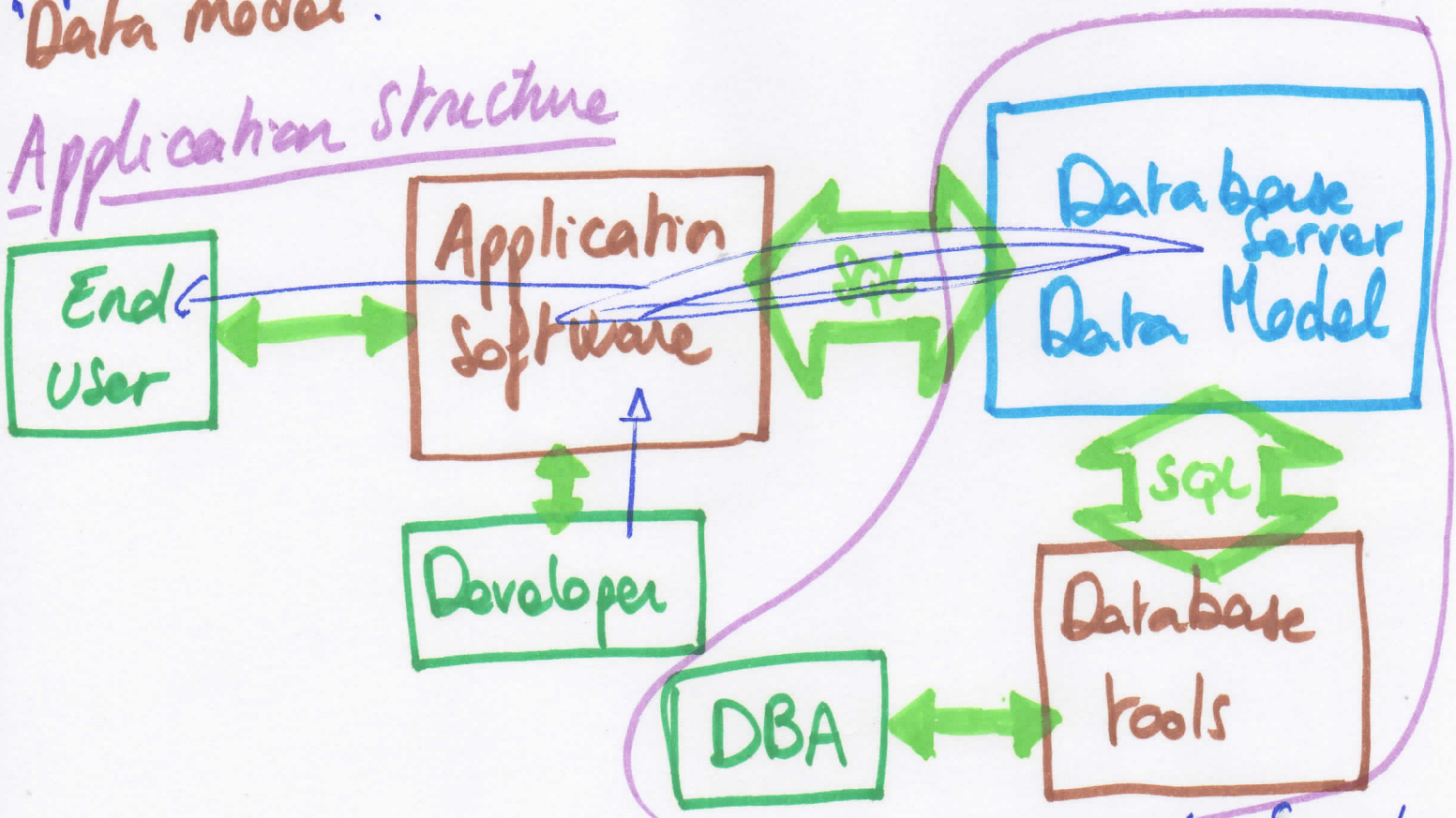
Two roles in large projects

① **Application developer**: builds the logic for the application, the look and feel of the application - monitors the application for problems.

② **Database administrator**: monitors and adjusts the database as the program runs in production.

- Often both people participate in the building of the "Data model".

Application structure



- The application developer writes code that talk SQL to the database and then get the data back and formate it for the end user. He creates application software. We don't usually let him talk directly to the database production in case he breaks it.
- The Database administrator is allowed to talk directly to the database

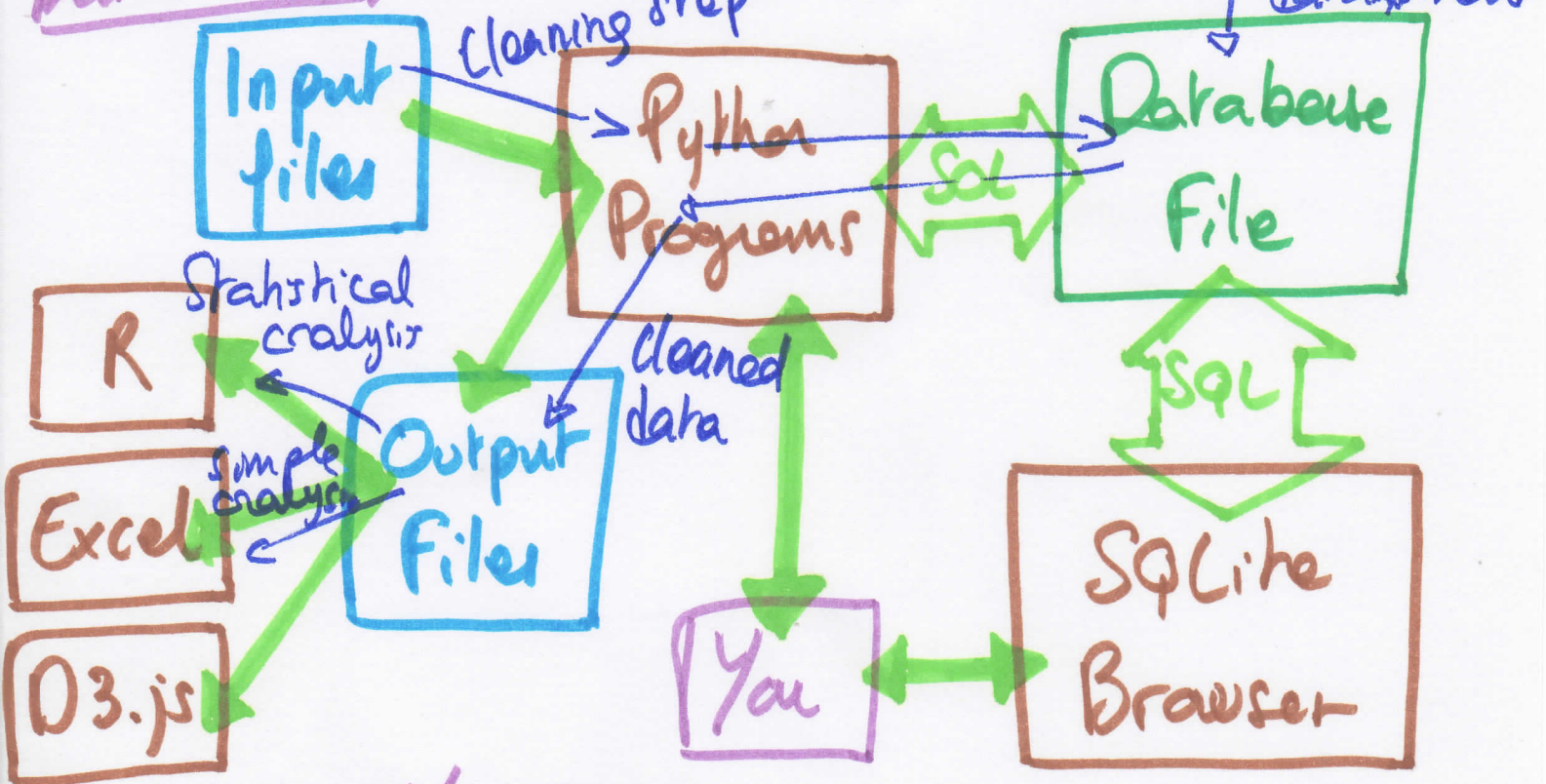
(71) Using databases with Python

Univ of Michigan chapter 14

Database administrator (DBA)

- A database administrator (DBA) is a person responsible for the design, implementation, maintenance, and repair of an organization's database. The role includes the development and design of database strategies, monitoring and improving database performance and capacity, and planning for future expansion requirements. They may also plan, coordinate, and implement security measures to safeguard the database.

Data Analysis Structure



Database Model

A database model or database schema is the structure or format of a database, described in a formal language supported by the database management system. In other words, a "database model" is the

(72) Using database with Python

Univ of Michigan chapter 14

application of a data model when used in conjunction with a database management system.

- It is like a **contract** about the shape of the data. (schema)

Common Database Systems

- Three major Database Management Systems in wide use:
  - **Oracle** - large, commercial, enterprise-scale, very very breakable → for online website mostly
  - **MySQL** - Simpler but very fast and scalable - commercial open source (acquired by Oracle) → very smaller
  - **SqlServer** - Very nice - from Microsoft (also Access)
- Many other smaller projects, free and open source
  - HSQL, **SQLite**, Postgres... → more like Oracle rich database

SQLite Database Manager

- It is an embedded database. It is built in bits of software: python, google, apple, mozilla, Adobe, Skype...
- It is a very popular database, free, fast and small
- There is a Firefox plugin to manipulate SQLite database!
- It is embedded in Python and a number of other languages
- It allows us to manipulate directly the SQLite files

(73) Using database with Python

Univ of Michigan chapter 14

Create a single table

```
CREATE TABLE Users (name VARCHAR(128),
email VARCHAR(128))
```

name of the table — up to 128 char. (size)

type of data (super characters)

This is the contract. (we cannot put super characters)
It allow the data to be very fast

Table created — Users

data inserted afterwards

name	email
Chuck	csev@umich.edu
Colleen	vit@umich.edu
Sally	sally@umich.edu
Fred	fred@umich.edu

SQL insert

- The insert statement inserts a row into a table

```
INSERT INTO Users (name, email) VALUES
('Ted', 'ted@umich.edu')
```

Ted	ted@umich.edu

SQL delete

- Deletes a row in a table based on a selection criteria

```
DELETE FROM Users WHERE email='ted@umich.edu'
```

(74) # Using databases with Python

Univ of Mich chapter 14

SQL update

- Allows the updating of a field with a where clause

```
UPDATE Users SET name = 'Charles' WHERE
email = 'csev@umich.edu'
```

Chuck Charles	csev@umich.edu

Retrieving records: select

- The select statement retrieves a group of records - you can either retrieve all the records or a subset of the records with a WHERE clause.

```
SELECT * FROM Users
        └─ all rows and columns

SELECT * FROM Users WHERE email = 'csev@
umich.edu'
```

Sorting with ORDER BY

- You can add an ORDER BY clause to SELECT statements to get the results sorted in ascending or descending order.

```
SELECT * FROM Users ORDER BY email

SELECT * FROM Users ORDER BY name
```

Database demo with Python (emaildb.py)

```
import sqlite3
```
) library to talk to the SQL database

(75) Using databases with Python

Univ of Michi
Chapter 14

```
conn = sqlite3.connect('emaildb.sqlite')
cur = conn.cursor()
```

we are gonna use a database in this application and store it in this file.

```
cur.execute('''
DROP TABLE IF EXISTS Counts''')
```

allow us to send commands

we ask for a cursor to get the commands

```
cur.execute('''CREATE TABLE Counts (email TEXT, count INTEGER)''')

fname = raw_input('Enter file name: ')
if (len(fname) < 1): fname = 'mbox-short.txt'
fh = open(fname)
for line in fh:
    if not line.startswith('From: '): continue
    pieces = line.split()
    email = pieces[1]
    cur.execute('SELECT count FROM Counts WHERE email = ?', (email,))
```

correspond

tuple

Email is what will be substitute for ?

placeholder to be filled in (with email)

```
    try:
        count = cur.fetchone()[0]
        cur.execute('UPDATE Counts SET count = count + 1 WHERE email = ?', (email,))
    except:
        cur.execute('''INSERT INTO Counts (email, count)
                VALUES (?, 1)''', (email,))
    conn.commit()
```

fetchone()[0] brings us one row into memory as a list

if there is no row it will blow up

Everything I've done until now, write it on the disk

(76) Using databases with Python

Univ of Michigan
Chapter 14

```
sqlstr = 'SELECT email, count FROM Counts ORDER BY
count DESC LIMIT 10'

print "Counts:"
for row in cur.execute(sqlstr):
    print str(row[0]), row[1]) it will be printed in the console
cur.close()
```

Database design

- Database design is an **art form** of its own with particular skills and experience.
- Our goal is to avoid the really bad mistakes and design clean and easily understood databases
- Others may performance tune things later
- Database design starts with a picture (think about BV)
- When you build an application, you need to design and create a database.
  - How we want to store our data in the database
- "Which tables are we going make? What will we put in the tables? How will we connect the tables?"
- The connections are what make the database powerful and fast

Building a data Model

- Drawing a picture of the data objects for our application and then figuring out how to represent the objects and their relationships

(77) Using database with Python — Uni of Michigan Chapter 14

- Basic rule: don't put the same string data in twice - use relationship instead
- When there is one thing in the "real world" there should be one copy of that thing in the database.
- Construct the data model in a way that you can use it to make the user interface the user wants.

For each "piece of info"...

- Is the column an object or an attribute of another object?
- Once we define objects, we need to define the relationships between objects.
- To start at the right place "what is the thing that is most essential/important to the application?" e.g. application that manages music tracks. we first build the "track" table. What is part of the track? We make a table with it.
- What is in the data model determines what the application is capable of.

Database Normalization (3NF)

- There is tons of database theory - way too much to understand without excessive predicate calculus. data
  - Do not replicate data - reference data - point at
  - Use integers for keys and for references
  - Add a special "key" column to each table which we will make references to. By convention, we call this column "id"

Using databases with Python

Univ of Michigan
Chapter 16

Integer Reference Pattern

Artist

id	artist
1	Ades
2	AC/DC

album

id	artist_id	title
1	2	Who
2	2	IV

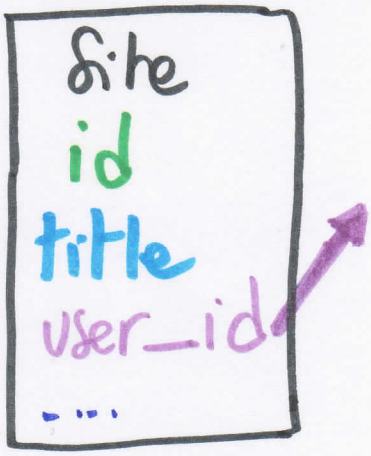
We use integers to reference rows in another table.

Three kinds of keys

Primary key: generally an integer auto-increment field

Logical key: what the outside world uses for lookup

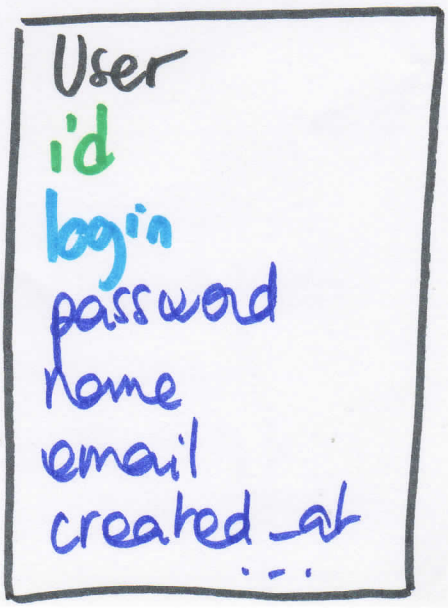
Foreign key: generally an integer key pointing to a row in another table.



Primary key rules

Best practices:

- Never use your **logical key** as the **primary key**
- **Logical keys** can and do change, albeit slowly
- **Relationships** that are based on matching string fields are less efficient than integers - performance-wise.

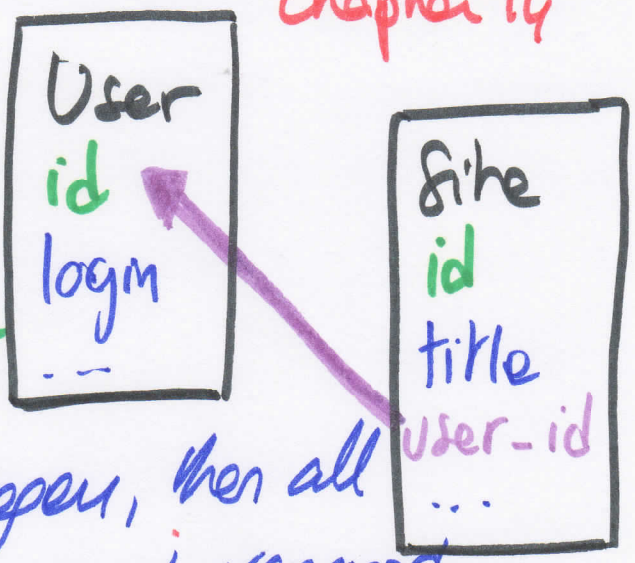


Using databases with PyMa

Univ of Michi
Chapter 14

Foreign Keys

- A foreign key is when a table has a column that contains a key which points to the primary key of another table.
- When all primary keys are integers, then all foreign keys are integers - this is good - very good
- If you use strings as foreign keys - NOT GOOD



Relationships building (in tables) Example

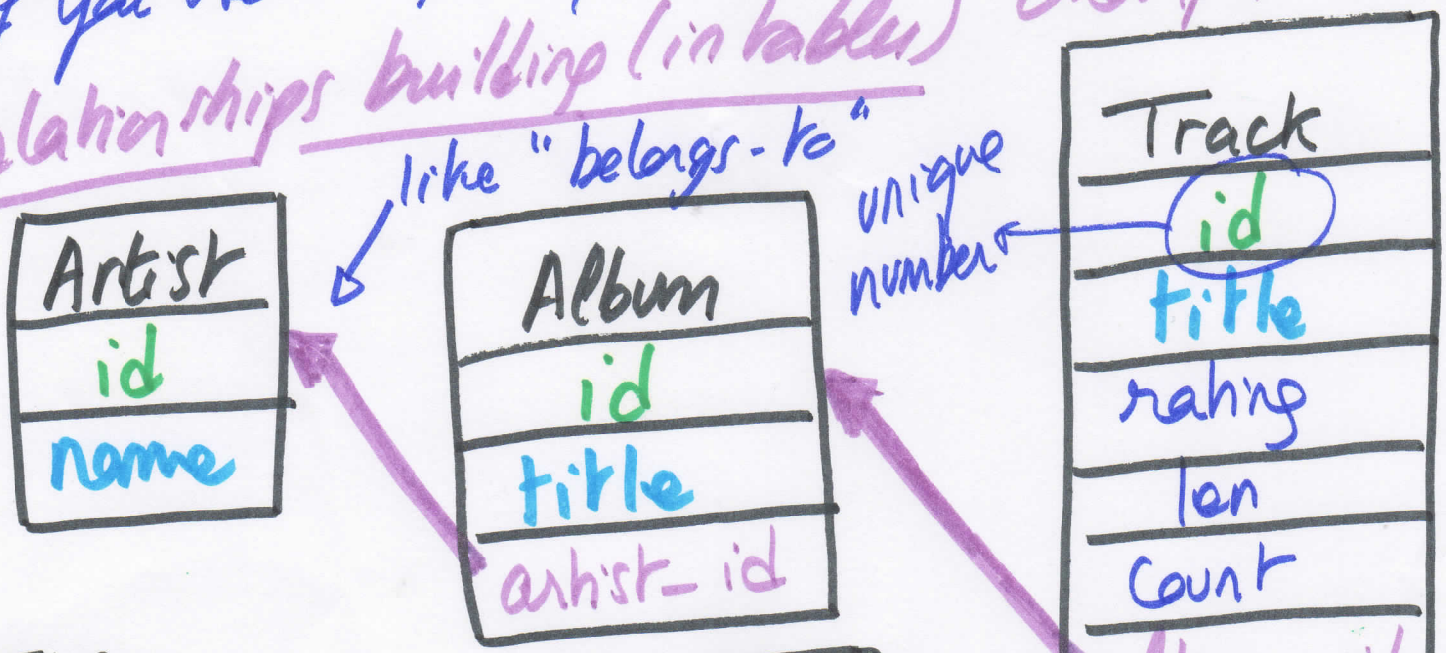


Table
Primary Key
Logical Key
Foreign Key

Naming FK artist_id is a convention. But it is not mandatory.

table name — id

(80) Using databases with Python

Univ. of Mich, chapter 14

Creating the tables - example

- We create the tables from outward in. ① Artist ② Genre ③ Album ④ Track
- For Track table:

```
CREATE TABLE Track (
    id INTEGER NOT NULL PRIMARY KEY
    AUTOINCREMENT UNIQUE,
    title TEXT,
    album_id INTEGER,
    genre_id INTEGER,
    len INTEGER, rating INTEGER, count INTEGER
```

Inserting relational data

- Example with first row of "Track" table

```
INSERT INTO Track (title, rating, len, count,
album_id, genre_id)
VALUES ('Black Dog', 5, 297, 0, 2, 1)
```

use the id from album and genre tables

- The primary keys (id) will be automatically created as said in the "contract" (when creating table)
- Using integers for primary keys (id) allows us to reduce enormously the amount of data to be scanned before getting a piece of data.

(81) Using database with Python — Univ of Michigan Chapter 14

Relationships between tables

Track

id	album_id	genre_id	len	rating	title	count
1	2	1	297	5	Black Dog	0
2	2	1	482	5	Stairc	0
3	1	2	313	5	Hello	0
4	1	2	207	5	Who	0

Genre

id	name
1	Rock
2	Metal

album

id	artist_id	title
1	2	Hey
2	1	IV

Artist

id	name
1	Led Zeppelin
2	AC/DC

There is no duplicated data

Relational power

- By removing the replicated data and replacing it with references to a single copy of each bit of data we build a "web" of information that the relational database can read through very quickly - even for very large amounts of data.
- Often when you want some data it comes from a number of tables linked by these foreign keys.

Using databases with Python

Univ of Michi
chapter 14

The JOIN operation

- The JOIN operation links across several tables as part of a select operation.
- You must tell the JOIN how to use the keys that make the connection between the tables using an ON clause.
- We want to connect one row and one table to the corresponding row in a different table.

Joining the tables

Album

id	artist_id	title
1	2	Who
2	1	IV

Artist

id	name
1	Led Zepplin
2	AC/DC

title	name
Who	AC/DC
IV	Led Zepplin

table name

column name

```
select Album.title, Artist.name from Album join Artist on Album.artist_id = Artist.id
```

What we want to see

The tables that hold the data

How the tables are linked.

on clause

⚠ Joining two tables without an ON clause gives all possible combinations of rows (which leads to duplicated data)

(83) Using databases with Python — Uni of Michigan chapter 14

- Then join Genre and Track
- Finally Track will the rest

```
Select Track.title, Artist.name, Album.title, Genre.
name from Track join Genre join Album join
Artist Track.genre_id = Genre.id and Track.al-
bum_id = Album.id and Album.artist_id =
Artist.id
```

Multi-table tracks done (≃ tracks.py) with Library.xml

(parts of the code)

create the table Artist only if it doesn't exist

```
cur.execute('''
CREATE TABLE IF NOT EXISTS Artist ( MENT UNIQUE
    id INTEGER NOT NULL PRIMARY KEY AUTOINCRE-
    name TEXT UNIQUE
)''')
```

→ logical key. Unique → cannot store two identical names in here

```
def lookup(d, key) :
    found = False
    for child in d :
        if found : return child.text
        if child.tag == 'key' and child.text == key :
            found = True
    return None
```

function that parse key/value pairs out of the XML dictionary object.

see later in code

```
name = lookup(entry, 'Name')
...
```

```
<dict>
  <key>Name</key> <string>who</string>
  <key>Total time</key> <integer>217</integer>
```

key tag

if found None, then go to the next tag (string) and return "who"

(84) Using database with Python

Uni of Michigan
Chapter 14

```
stuff = ET.parse(fname)          # read fname and we get the stuff object
all = stuff.findall('dict/dict/dict')
print 'Dict count:', len(all)
for entry in all:                # element in dict
    if (lookup(entry, 'Track ID') is None): continue   # skipping empty tracks

    name = lookup(entry, 'Name')
    artist = lookup(entry, 'Artist')      # for lookup function
    :
    if name is None or artist is None or album is None:
        continue

    print name, artist, album, ...

cur.execute('''INSERT OR IGNORE INTO Artist (name)
    VALUES (?)''', (artist,))
cur.execute('SELECT id FROM Artist WHERE name = ?',
    (artist,))
artist_id = cur.fetchone()[0]

cur.execute('''INSERT OR REPLACE INTO Track
    (title, album_id, len, rating, count)
    VALUES (?, ?, ?, ?, ?)''',
    (None, album_id, length, rating, count))
```

- `stuff = ET.parse(fname)` — read fname and we get the stuff object
- `all = stuff.findall('dict/dict/dict')`
- `for entry in all:` — element in dict
- `if (lookup(entry, 'Track ID') is None): continue` — skipping empty tracks
- `name = lookup(entry, 'Name')` / `artist = lookup(entry, 'Artist')` — for lookup function
- `if name is None or artist is None or album is None: continue`
- `print name, artist, album, ...`
- `cur.execute('''INSERT OR IGNORE INTO Artist (name) VALUES (?)''', (artist,))` — from the CREATE statement take advantage of the constraint (UNIQUE). Insert if it's not already there or ignore
- `cur.execute('SELECT id FROM Artist WHERE name = ?', (artist,))` — we put this id into artist_id
- `artist_id = cur.fetchone()[0]`
- `cur.execute('''INSERT OR REPLACE INTO Track (title, album_id, len, rating, count) VALUES (?, ?, ?, ?, ?)''', (None, album_id, length, rating, count))` — combination of INSERT / UPDATE kind of. Several set into one 5-tuple
- if the title doesn't exist, insert the row. If it does, update all the rest

Using databases with Python

Univ of Michi
Chapter 14

Complexity enables speed

- Complexity makes speed possible and allows you to get very fast results as the data size grows.
- By **normalizing the data** and **linking it with integer keys**, the overall **amount of data** which the relational database must scan is **far lower** than if the data were simply flattened out.
- It might seem like a **tradeoff** - spend some time designing your database so it continues to be fast when your application is a success.

Additional SQL topics

- **Indexes**: improve access performance for things like string fields
- **Constraints** on data - (cannot be NULL, etc...)
- **Transactions** allow SQL operations to be grouped and done as a unit
- **See S1664 - Database Design**

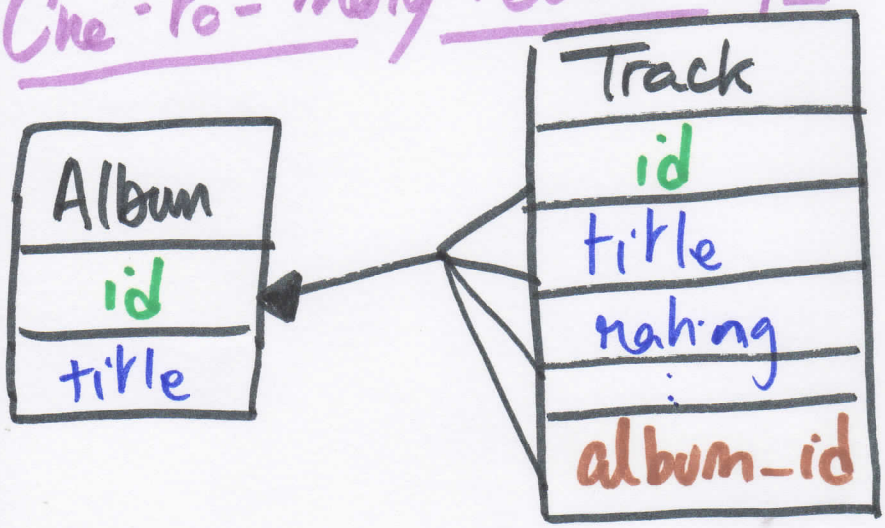
Summary - Databases

- Relational databases allow us to **scale** to very large amounts of data
- The key is to have **one copy of any data** element and use relations and joins to link the data to multiple places.

Using databases with Python

- This greatly reduces the amount of data which much be scanned when doing complex operations across large amounts of data
- Database and SQL design is a bit of an art form.

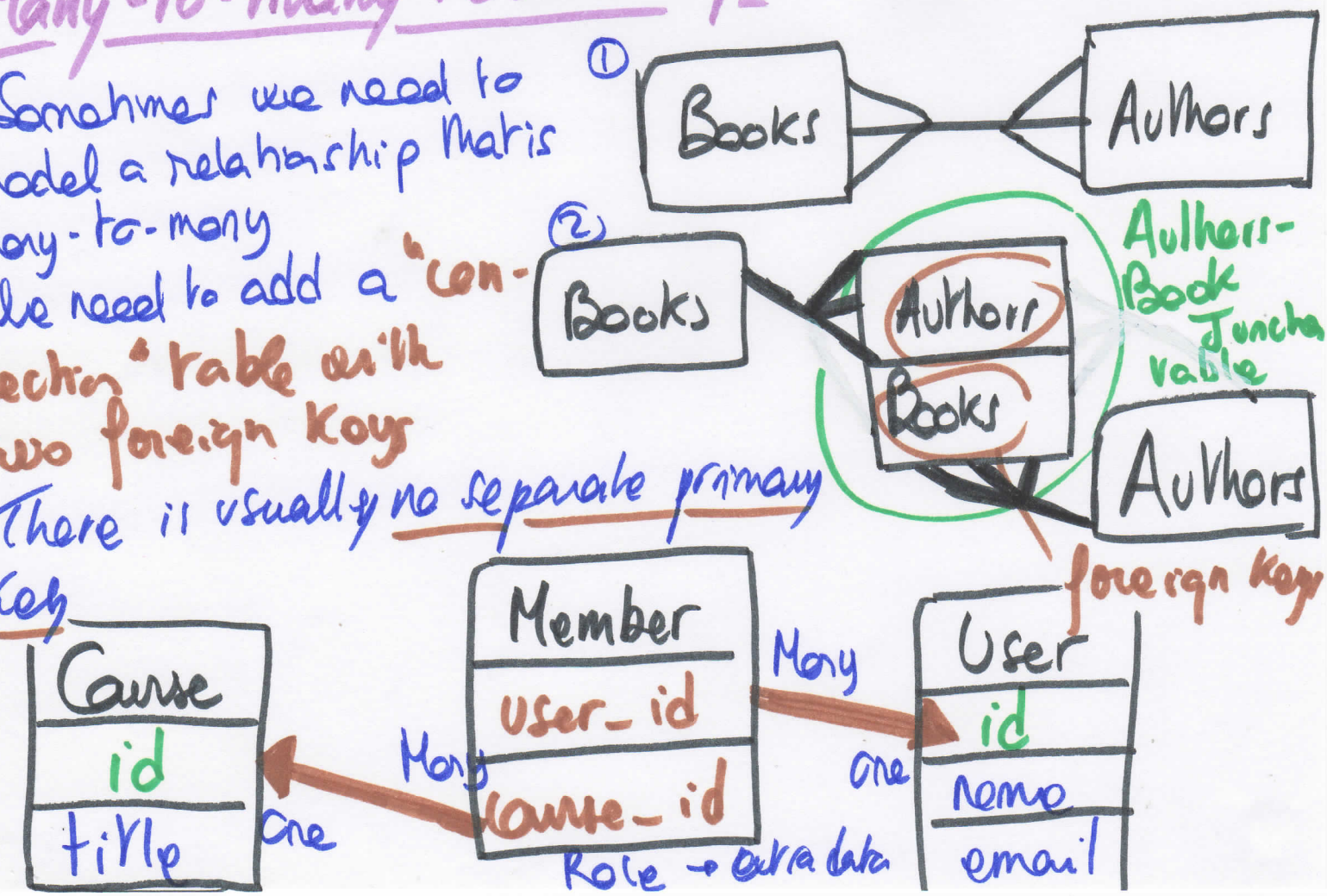
One-to-many relationship



In one album, there are many tracks.

Many-to-many relationship

- Sometimes we need to model a relationship that is many-to-many
- We need to add a "connection" table with two foreign keys
- There is usually no separate primary key



(87) Using database with Python — Univ of Michigan chapter 14

Create a database with many-to-many relationships:

①
```
CREATE TABLE User (
    id      INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
    name    TEXT,
    email   TEXT )

CREATE TABLE Course(
    id      INTEGER NOT NULL PRIMARY KEY AUTO... )
    title   TEXT )
```

connector / junction table

```
CREATE TABLE Member(
    user_id     INTEGER
    course_id   INTEGER
    role        INTEGER,
    PRIMARY KEY (user_id, course_id) )
```

same extra data allowing us to tell if teacher or student

② Insert data into User and Course tables

③ Insert data into Member table

user 1 — course 1 — make 1 the teacher

```
INSERT INTO Member (user_id, course_id, role)
    VALUES (1, 1, 1);
```

0 will be student

```
INSERT INTO Member (user_id, course_id, role)
    VALUES (2, 1, 0);
INSERT INTO Member (user_id, course_id, role)
    VALUES (3, 3, 0)
```

(87) Using database with Python — Univ of Michigan chapter 14

Create a database with many-to-many relationships:

1)
```
CREATE TABLE User (
    id      INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE,
    name    TEXT,
    email   TEXT )

CREATE TABLE Course(
    id      INTEGER NOT NULL PRIMARY KEY AUTO... )
    title   TEXT )
```

connector / junction table

```
CREATE TABLE Member(
    user_id   INTEGER
    course_id INTEGER
    role      INTEGER,
    PRIMARY KEY (user_id, course_id) )
```

some extra data allowing us to tell if teacher or student

② Insert data into User and Course tables

③ Insert data into Member table

user 1 — course 1 — make 1 the teacher

```
INSERT INTO Member (user_id, course_id, role)
    VALUES (1, 1, 1);
```

0 will be student

```
INSERT INTO Member (user_id, course_id, role)
    VALUES (2, 1, 0);
        ⋮
INSERT INTO Member (user_id, course_id, role)
    VALUES (3, 3, 0)
```

(88) Using database with Python

Uni of Michi chapter 14

id	nome	email
1	Jane	jane@email
2	Ed	ed@email
:	:	:

User

user_id	course_id	role
1	1	1
2	1	0
3	1	0
:	:	:

Member

No primary Key

	nome	role	title
2	Sue	0	PHP
3	Jane	1	Python
4	Ed	0	Python
:	:	:	:

Course

id	title
1	Python
2	SQL
:	:

```
SELECT User.nome, Member.role, Course.title
FROM User JOIN Member JOIN Course
ON Member.user_id = User.id AND Member.course_id
                  = Course.id
ORDER BY Course.title, Member.role DESC, User.nome
```

most important

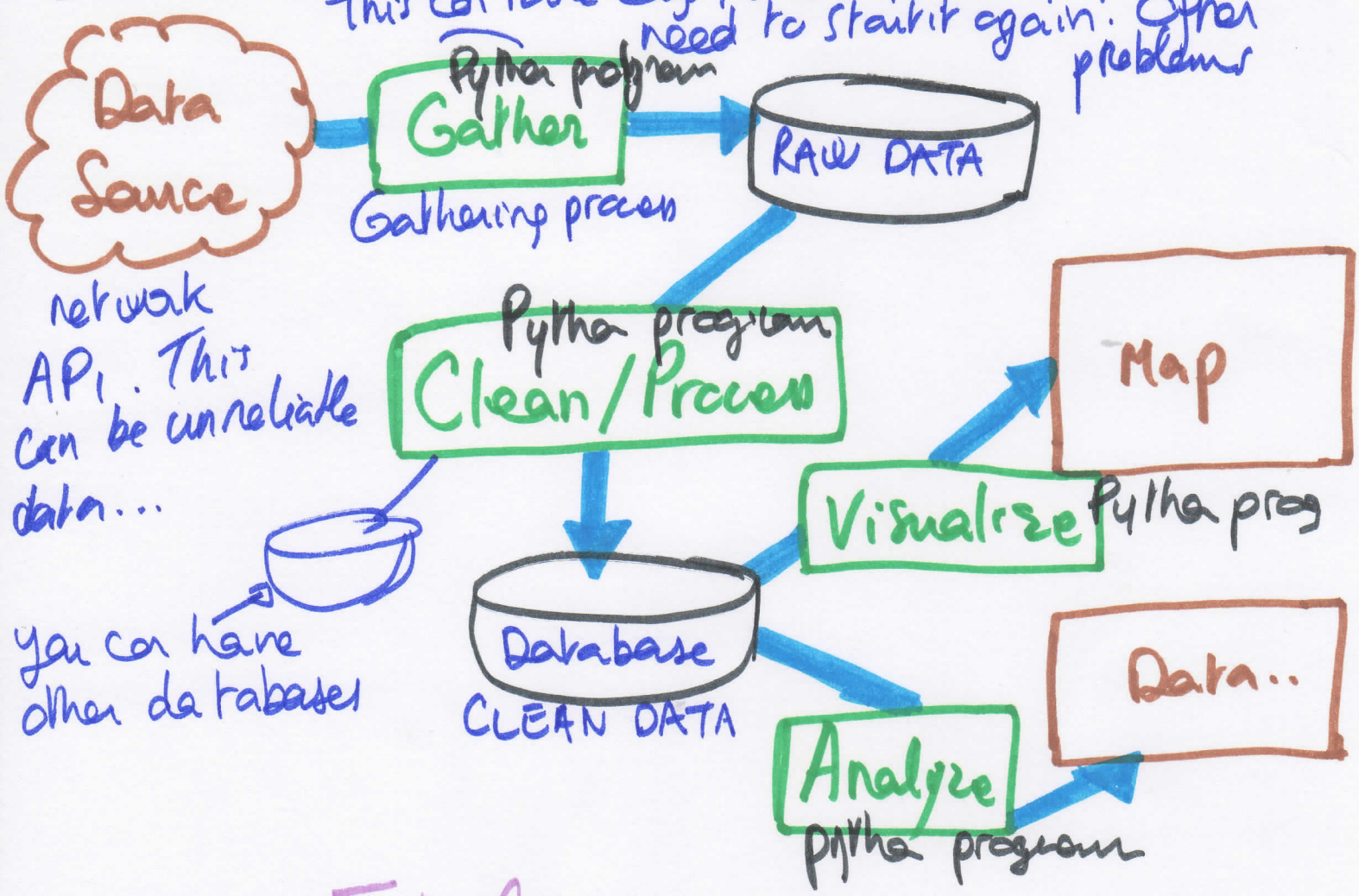
---

cur.executescript() => we can execute multiple SQL commands

Using databases with Python

Univ of Mich chapter 15

Multi-step data analysis



Data Mining Technologies

- This course is not exactly about data mining but it covers some aspects (little). It is a particular style of data mining.
  - https://hadoop.apache.org
  - http://spark.apache.org
  - https://aws.amazon.com/redshift
  - http://community.pentaho.com ...

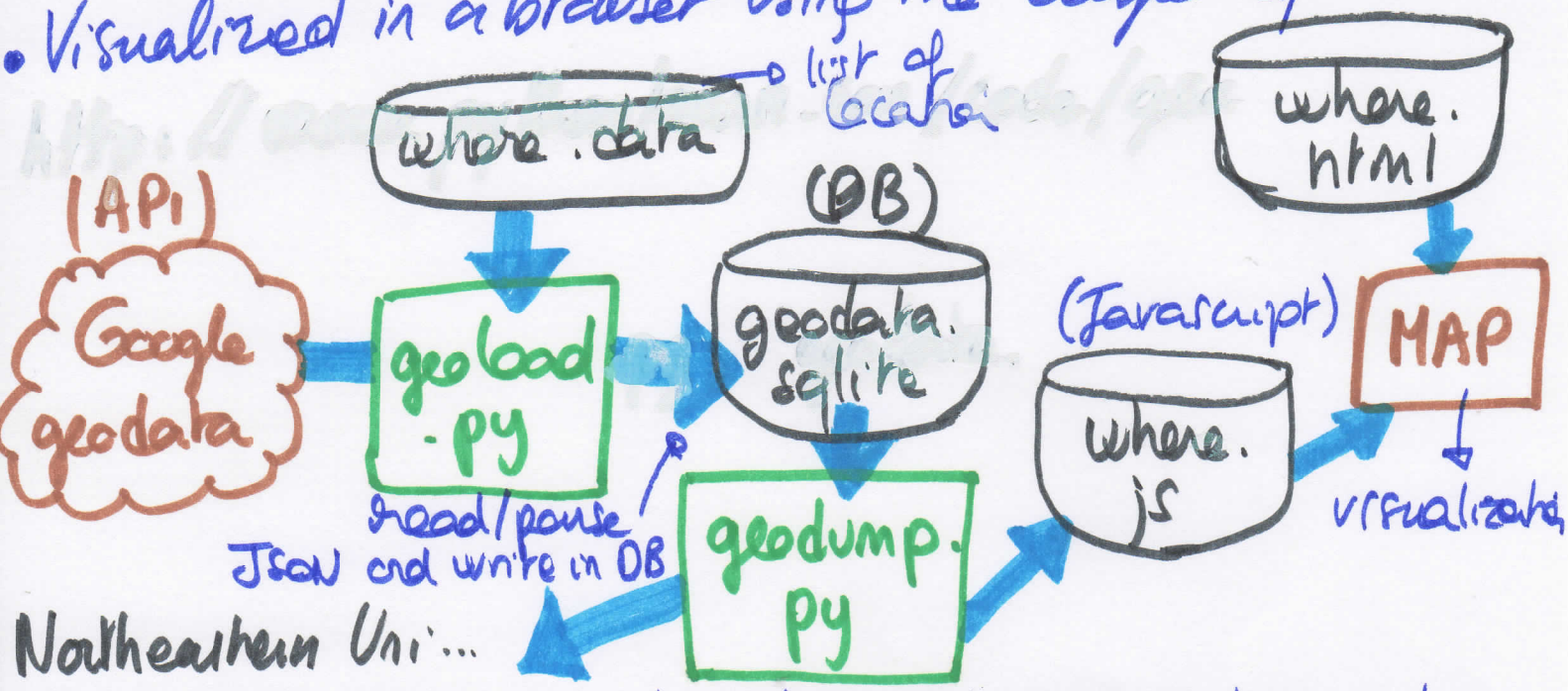
Geo Data

- Makes a Google Map from user entered data

(90) Using database with Python

Uni of Michigan
chapter 15

- Uses the Google Geodata API
- Caches data in a database to avoid rate limiting and allow restarting
- Visualized in a browser using the Google Maps API



Northeastern Uni...
USA 42.33 -71.08 ...
(data)
12 records written to where.js

loop through the DB, print some data and write a where.js file.

Page Rank

- Write a simple web page crawler
- Compute a simple version of Google's Page Rank algorithm
- Visualize the resulting network

Search Engine Architecture

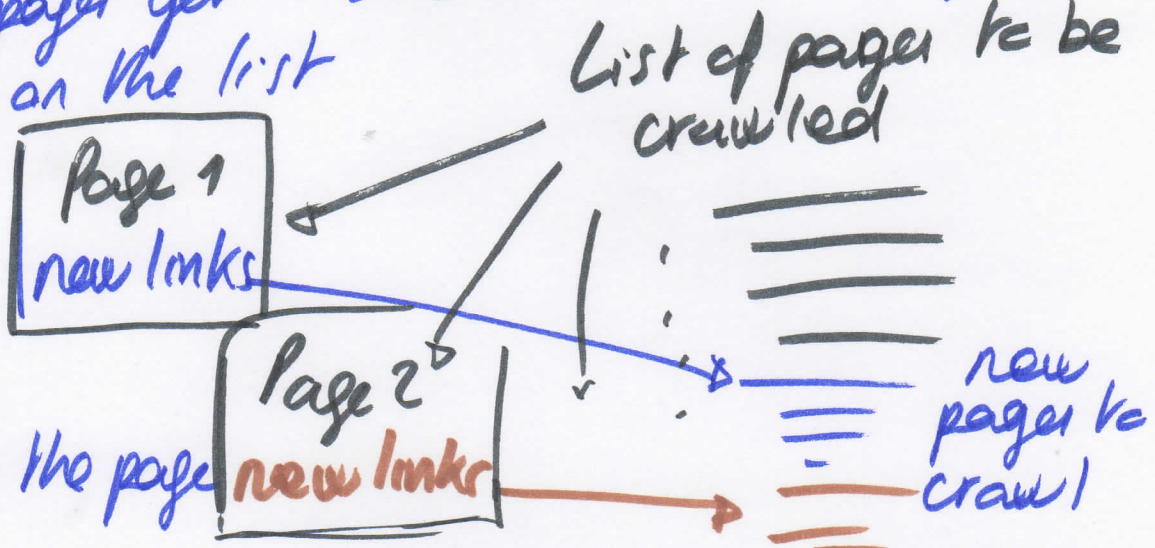
1. Web crawling - gathering phase — we crawl
2. Index building - page rank - Some making of the data that
3. Searching - looking at the data and showing the right pages. -> (we will do visualization)

(91) Using databases with Python

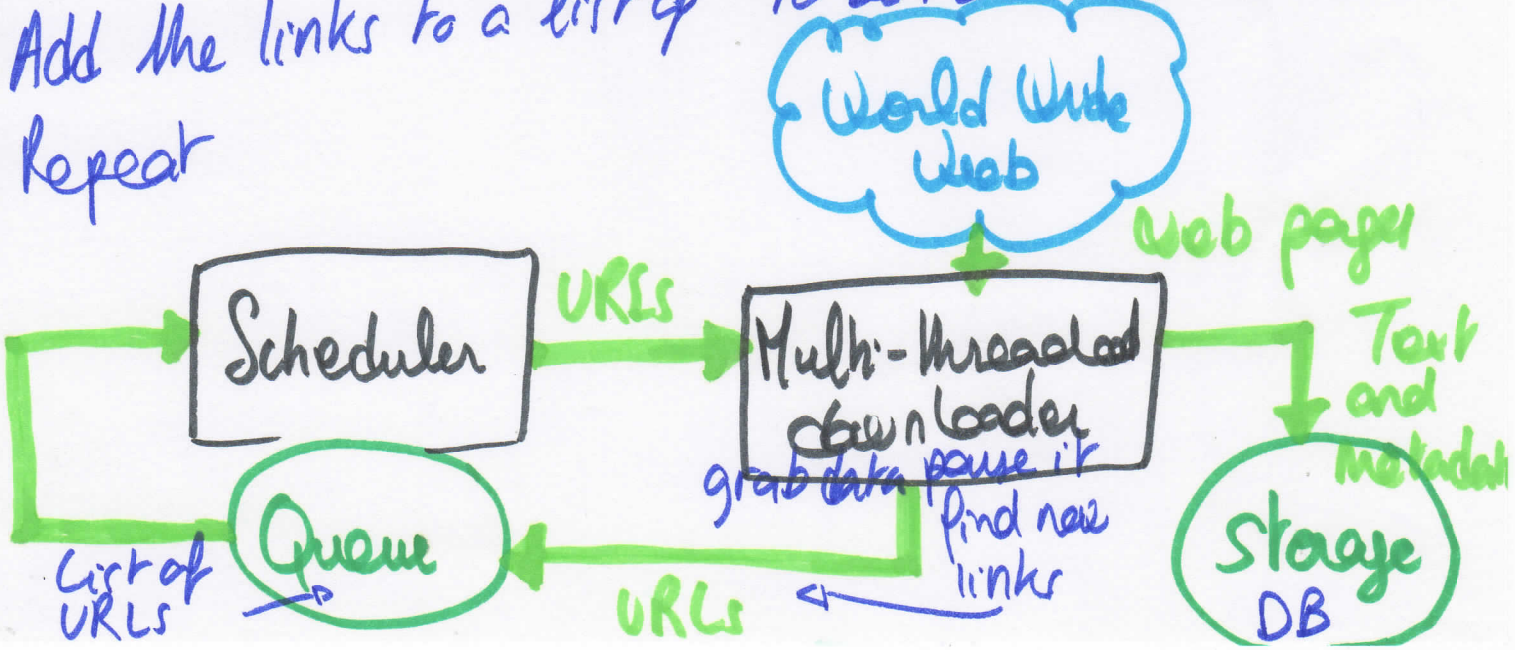
Uni of Michigan
chapter 15

Web crawler

- A web crawler is a computer program that browses the World Wide Web in a methodical, automated manner. Web crawlers are mainly used to create a copy of all the visited pages for later processing by a search engine that will index the downloaded pages to provide fast searches.
- Piece of software that gives itself a list of pages to look at, read those pages, use an algorithm like beautiful soup to extract the links from the pages, add those links to pages yet to be crawled and then go grab the next page on the list



1. Retrieve a page
2. Look through the page for links
3. Add the links to a list of "to be retrieved" files
4. Repeat



Using database with PyMa

Univ of Michigan
chapter 15

Web crawling policy

- A selection policy that states which pages to download
- A re-visit policy that states when to check for changes to the pages
- A politeness policy that states how to avoid overloading Web sites.
- A parallelization policy that states how to coordinate distributed Web crawlers.

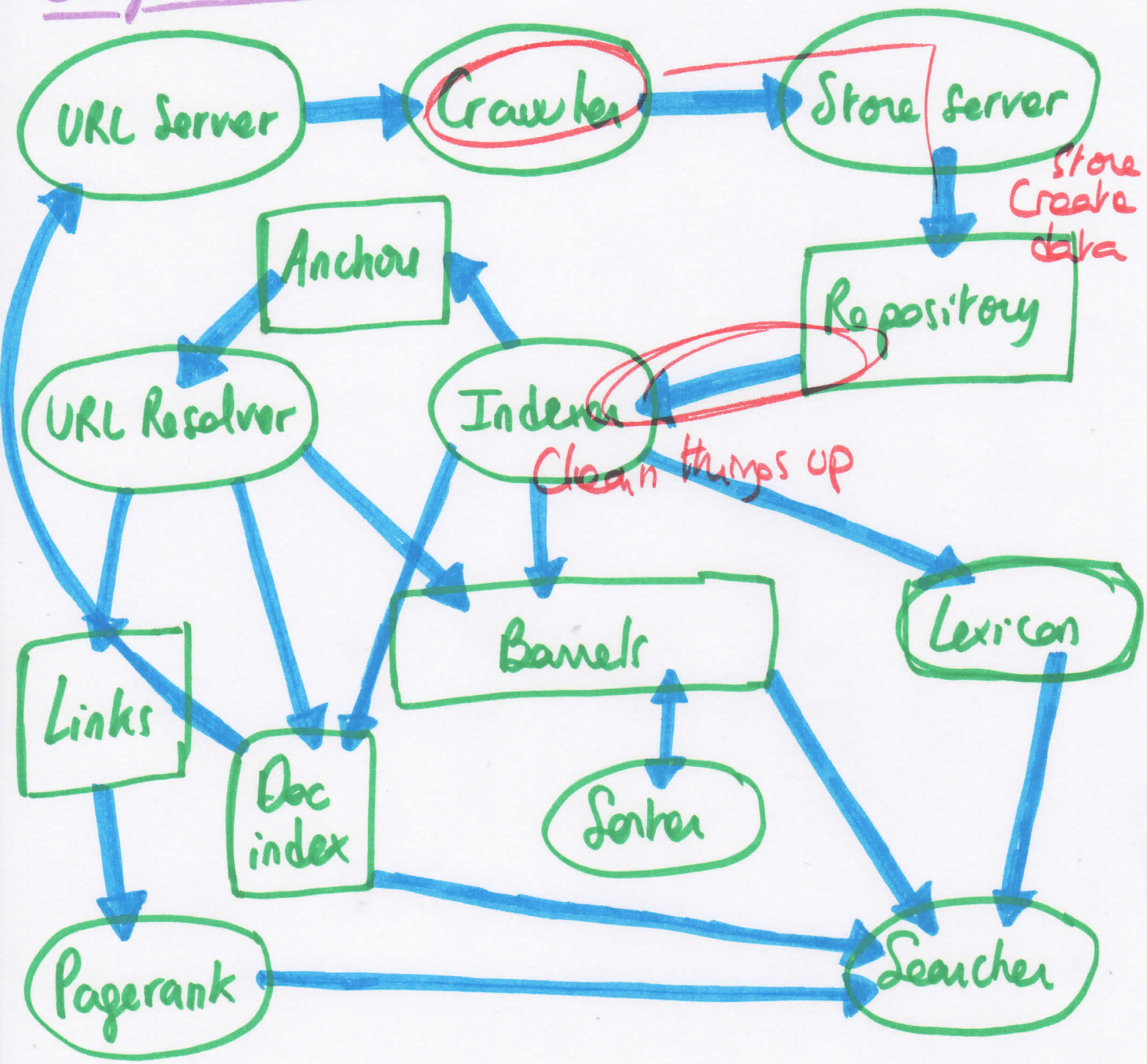
robots.txt

- A way for a web site to communicate with web crawlers
- An informal and voluntary standard
- Sometimes folks make a "Spider Trap" to catch "bad" spiders
  → you put the code at the top of the website.

```
User-agent: *
Disallow: /cgi-bin/
Disallow: /images/
Disallow: /tmp/
Disallow: /private/
```

Using databases with Python

Google architecture



Search indexing

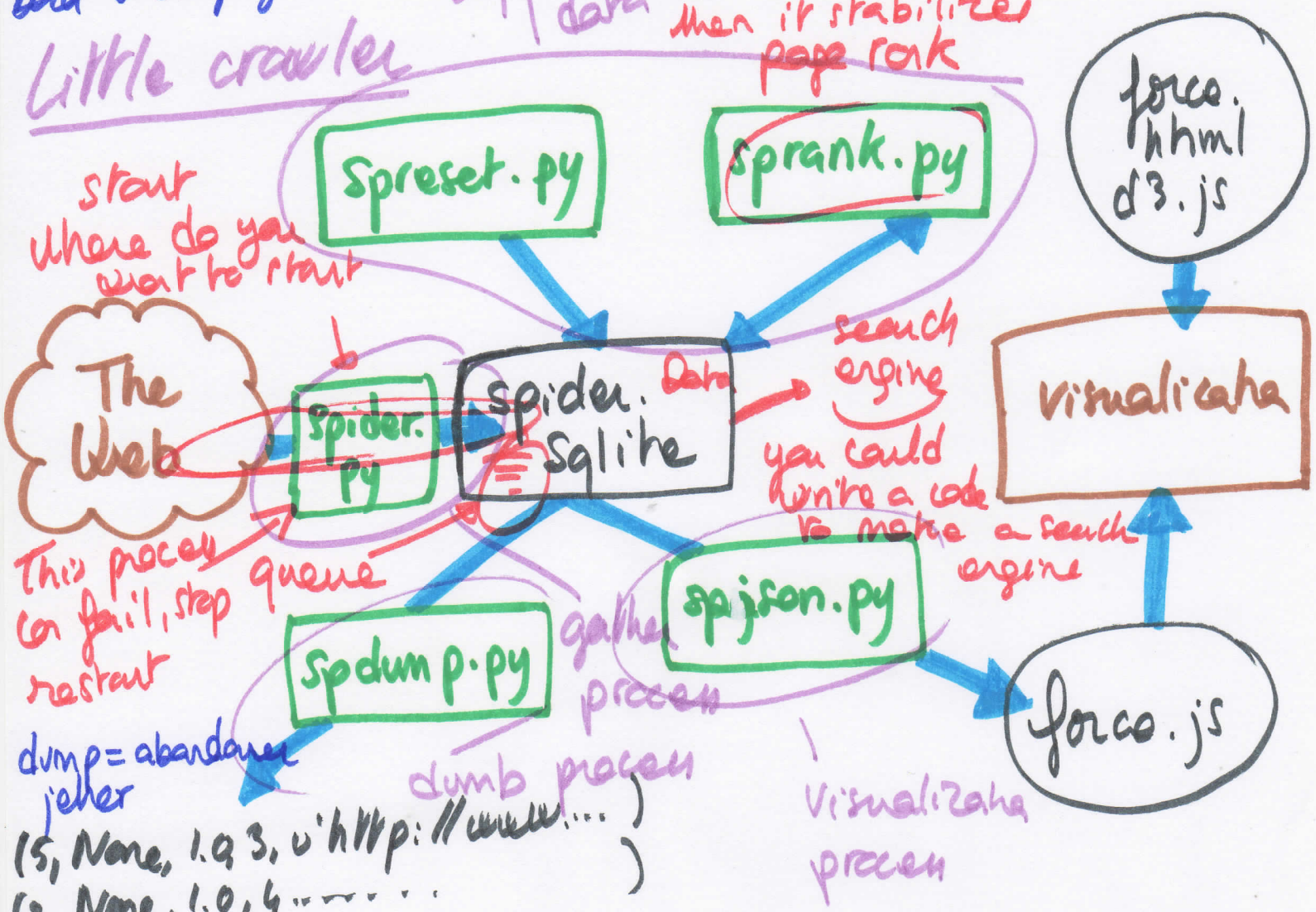
- Search engine indexing collects, parses, and stores data to facilitate fast and accurate information retrieval. The purpose of storing an index is to optimize speed and performance in finding relevant documents for a search query. Without an index, the search engine would scan

Using databases with Python

every document in the corpus, which would require considerable time and computing power.

- It looks for useful meaning inside web pages. (and if it's bad web pages)

Little crawler

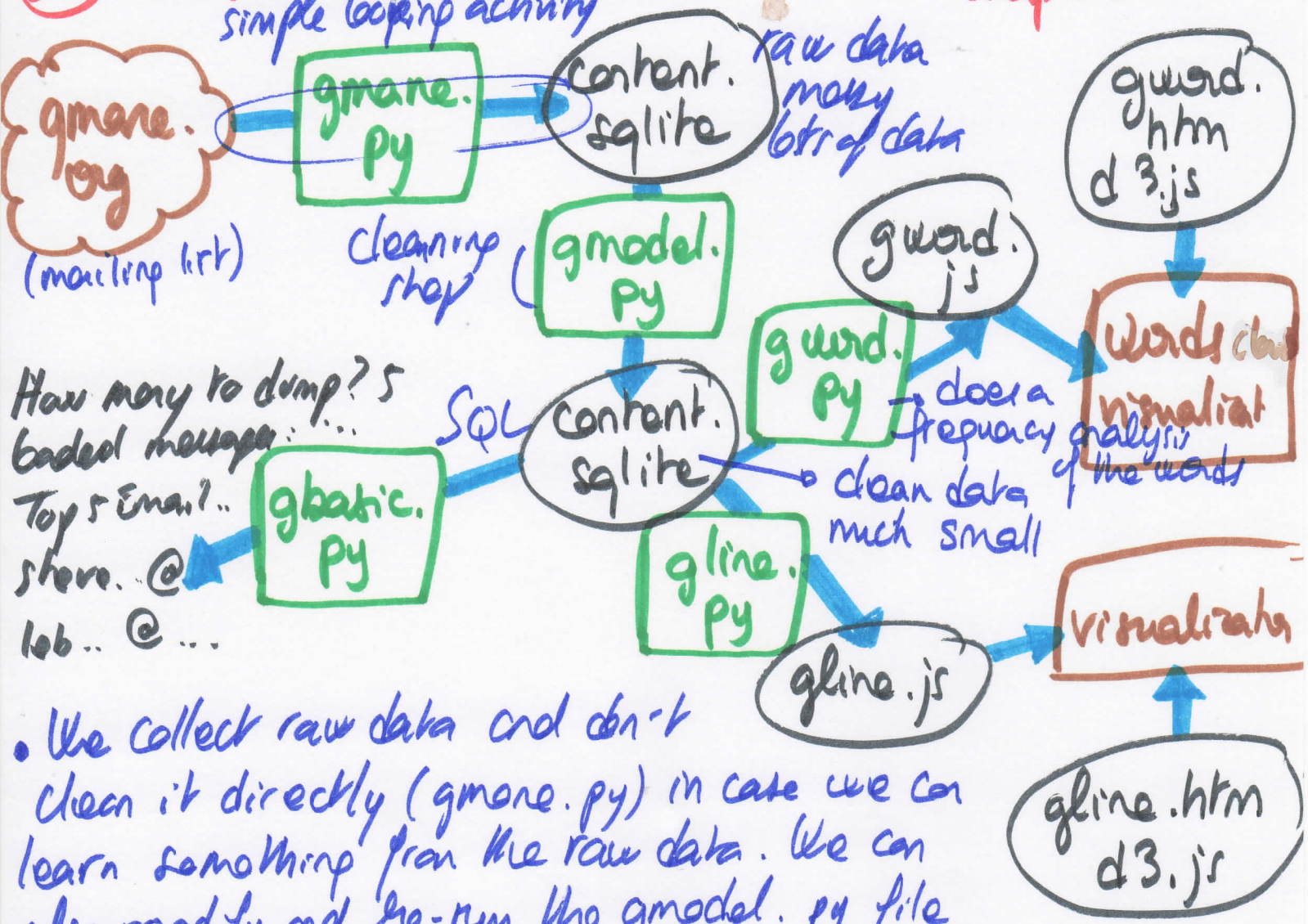


Mailing lists - Gmane

- Archive mailing list
- Crawl the archive of a mailing list
- Do some analysis/cleanup
- Visualize the data as word cloud and line

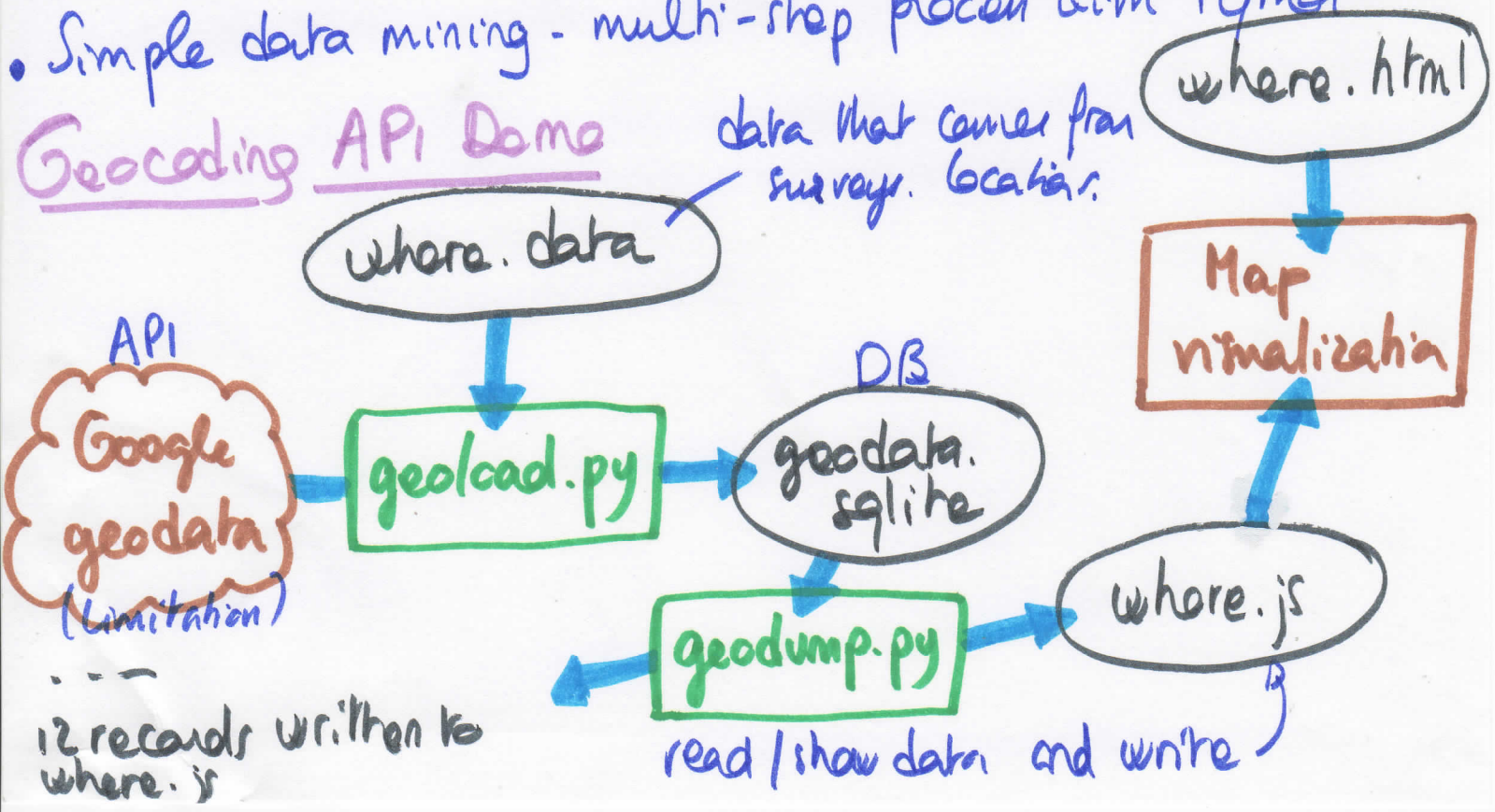
(95) Using database with Python

Univ of Michigan chapter 15



- We collect raw data and don't clean it directly (gmane.py) in case we can learn something from the raw data. We can also modify and re-run the gmodel.py file on the raw data
- Simple data mining - multi-step process with Python

Geocoding API Demo



Using databases with Python

except:
pass] break the loop and continue to the next line of code (but don't quit)

CTRL + Z to stop the program running on the console